



Original citation:

Alexander-Craig, I. D. (1987) A distributed Blackboard architecture. University of Warwick. Department of Computer Science. (Department of Computer Science Research Report). (Unpublished) CS-RR-091

Permanent WRAP url:

<http://wrap.warwick.ac.uk/60787>

Copyright and reuse:

The Warwick Research Archive Portal (WRAP) makes this work by researchers of the University of Warwick available open access under the following conditions. Copyright © and all moral rights to the version of the paper presented here belong to the individual author(s) and/or other copyright owners. To the extent reasonable and practicable the material made available in WRAP has been checked for eligibility before being made available.

Copies of full items can be used for personal research or study, educational, or not-for-profit purposes without prior permission or charge. Provided that the authors, title and full bibliographic details are credited, a hyperlink and/or URL is given for the original metadata page and the content is not changed in any way.

A note on versions:

The version presented in WRAP is the published version or, version of record, and may be cited as it appears here. For more information, please contact the WRAP Team at: publications@warwick.ac.uk



<http://wrap.warwick.ac.uk/>

Research report 91

A DISTRIBUTED BLACKBOARD ARCHITECTURE

Iain D Craig*

(RR91)

Abstract

This paper describes a problem solving architecture derived from the blackboard model of problem solving. The architecture introduces the concepts of Level Managers and local control into the blackboard architecture. The architecture is suited to distributed problem solving. The major features of the architecture are described and the restrictions imposed upon components by an inherently distributed, asynchronous problem solving environment are discussed. An experimental system, called CASSANDRA-II, is described, as is the control component derived in an exercise to re-design HASP/SIAP in terms of the new architecture. The CASSANDRA-II system implements the features of the architecture in the context of intelligent monitoring of aircraft movements in controlled airspace.

* The work reported in this paper was conducted while the author was with the Department of Computing, University of Lancaster, Lancaster LA1 4YR.

Department of Computer Science
University of Warwick
Coventry
CV4 7AL, UK

Jan 1987

1. INTRODUCTION

The blackboard model of problem solving ([Balzer, 1980],[Craig, 1986], [Erman, 1975], [Erman, 1981], [Hayes-Roth, 1979], [Hayes-Roth, 1983], [Hayes-Roth, 1984], [Hayes-Roth, 1984b], [Hayes-Roth, 1985a], [Hayes-Roth, 1985b], [Hayes-Roth, 1986], [Nii, 1979], [Nii, 1986a], [Nii, 1986b], [Terry, 1983]) has been successfully applied to a number of domains. As the size and complexity of problem increases, there is an argument for exploiting potential concurrency in the blackboard model. Many blackboard systems have been constructed to solve problems in real time domains where reliability is also a major issue.

This paper describes a system, called CASSANDRA-II, and the architecture which underlies the system. The architecture, which will be referred to as the CASSANDRA architecture, is an extension of the conventional blackboard architecture. It provides a natural framework for the construction of distributed problem solving systems. The CASSANDRA-II system monitors aircraft movements in a simulated airspace and warns the user of any violations of air traffic control rules. Air traffic control is a real time domain in which reliability is absolutely necessary.

The paper is structured as follows. In the next section, the CASSANDRA architecture is briefly outlined. The description centres on the more novel aspects of the architecture. In section 3, an application system, the CASSANDRA-II airspace monitoring program is described.

Section 3 concentrates on the general architecture of the system. Section 4 briefly describes a design study in which the HASP/SIAP system ([Nii, 1986b]; [Feigenbaum, 1983]; [Feigenbaum, 1978]) was translated into the CASSANDRA architecture. Section 5 is by way of a conclusion.

2. THE ARCHITECTURE

This section presents a brief outline of the CASSANDRA architecture. The interested reader should consult ([Craig, 1987], ch. 3) for a more detailed exposition.

The CASSANDRA architecture is based on the conventional blackboard architecture. It differs most importantly in its provision of Level Managers (LMs). Level Managers represent a complete blackboard abstraction level: that is, they contain the solution elements (or entries) which represent the state of the problem solution on that level. The entries present in a LM are wholly local to that Level Manager. It is not possible for any LM to directly inspect another's entries. If one LM needs access to the entries held by another LM, it must explicitly make a request for that information.

Each LM contains a local database of entries. It also contains a set of Knowledge Sources (KSs), a matcher, an action interpreter and a local scheduler. The KSs within a LM inspect the LM's entries. They do this using the local matcher. When a KS precondition evaluates to true, it is placed in the local control database. Control databases are

wholly local to each LM and may not be directly inspected or updated by any other LM.

When a KS instance has been selected for execution in a LM, the Level Manager's action interpreter is used to run the KS action. The local matcher and action interpreter permit different representations of KSs to be present in one CASSANDRA system.

Control within a Level Manager is performed by the local controller. Each LM has its own controller and each local controller has access to a control database. Control is localised in a CASSANDRA system so that each LM may have the control regime it needs.

Local controllers cannot perform any global control decisions: instead, there is a global controller which is charged with implementing control decisions for the entire system. In a conventional blackboard system, the needs of individual abstraction levels are mixed with the global control properties required by the problem. In a CASSANDRA system, local and global control are separated. This factoring has a methodological advantage: it is possible to specify local and global control independently. It is also possible to change some local control structures while maintaining the overall functionality of the system.

Level Managers in a CASSANDRA system communicate by sending messages. Messages may only be sent along pre-defined communication channels. Each channel is associated with two ports: one for input, the other for output.

Information is put onto a channel by a sending LM and removed from by a receiving LM. Only two LMs may be associated with a channel: the CASSANDRA architecture does not allow broadcast messages to be sent by LMs.

LMs in a system built using the architecture can only communicate via uni-directional channels. Channels are connected to LMs via ports. Channels are motivated by the need for inter-LM communication in a system which is at least conceptually distributed. For each channel known to a Level Manager, there must be one and only one port. Ports are used for input or output and are strongly typed in the sense that any port can be used only for input or for output at any one time: it is not possible for a port to be used for both input and output at the same time. Ports have to be declared, as do channels. When a port is declared, its type (either input or output) is specified. When ports are declared, they are also connected to their associated channels. This permits checks to be made to ensure that the semantics of ports and channels is being enforced.

Since the Level Managers act as black boxes to the other components of a CASSANDRA system, ports and channels are the only way in which communication between Level Managers can be effected. Channels serve to enforce the distinctions between Level Managers in a system. If two LMs are not connected by a channel, they cannot communicate. The direction in which information flows along a channel also serves to define the relationship between the Level Managers in a system.

Channels are used to communicate a variety of information across a system. The information is usually in the form of individual blackboard entries or sets of entries. Knowledge Sources in each LM can send entries to those other LMs which are attached by channels.

Because the primary role of a Knowledge Source is to alter the blackboard state, channels must transmit entries. In some cases, KSs will need to perform a modification operation on the entries in some other LM. It is not possible to perform such modifications directly, so a control message, instead, is sent to the LM in which the entry resides. The control message informs the receiving LM that a modification is to be made, the type of the modification (add attribute, replace value in attribute, delete attribute, and so on) and the value (if any) which the attribute is to take on. Messages such as these may contain change information for sets of attributes. Such control messages can be conditionalised so that the receiving Level Manager can search for particular entries which satisfy the message's condition. This is because of the prohibition on the inspection of the local LM by external agents.

Control messages can also be sent by the global controller in a CASSANDRA system. Global control messages are sent to co-ordinate the problem solving activities of the LM system. Apart from the modification and link-chasing messages, global control messages are the only other class of control message allowed by the architecture. It is not permitted for one Level Manager to instruct another to

perform some action: if it were, the instruction would have to be based upon knowledge of the state of the LM receiving the instruction. The global controller, on the other hand, can request state information from each LM in the system or arrange for LMs to perform certain control actions upon receipt of a global control message. The variety of global control is not restricted in the definition of the CASSANDRA architecture: it remains, however, the only part of the architecture which may gather global information. Global control can only, really, make recommendations about what each LM should do next. This is because, in a distributed implementation, the information upon which the global controller makes its decisions may be out of date. The architecture imposes no requirements that information held by global controllers be always immediately up-to-date: it requires only that global control information be the best available.

In concurrent programs and problem solving systems, deadlock is a real threat. This fact applies also to the CASSANDRA architecture. There are two ways in which deadlock may enter into a system built using the architecture: in the channels between Level Managers and in the Knowledge Sources. It is not possible to remove the possibility of deadlock from channel configurations because each individual application will have different channel configurations and these cannot all be foreseen. Instead, the architecture makes restrictions on the behaviours of KSs.

In the architecture, KSs trigger on the contents of the entry database local to the Level Manager in which the KSs reside. KS actions may alter the state of the local database or they may cause alterations to the state of some other Level Manager's database. The architecture requires that KSs trigger on entries already in the local database: it is not possible for a KS to wait to trigger on messages coming into the LM from a channel. If this were permitted, there would be a large number of KS instances in a waiting state for it is not possible to assume any minimum transmission time for messages.

Similarly, KS preconditions[2] are restricted to examining entries in the local database. There is one exception to this: link-chasing. It is sometimes necessary for a KS to follow the links between entries on the blackboard. Link-chasing is used, for example, to gather information from a solution island. Given the distributed nature of the CASSANDRA blackboard and the restrictions imposed by the architecture, link-chasing would appear to be ruled out. Since link-chasing can be an important feature of an application, the architecture provides special facilities to perform link-chasing in as safe a fashion as possible.

Link-chasing is performed by making a request to the Level Manager in which the KS resides. The request specifies

[2] KSs in CASSANDRA systems have a trigger and a precondition -- the trigger is intended to be a quick check on the contents of the local database, whereas the precondition is intended to be a more costly state predicate.

one or more links and the names of the Level Managers which contain the entries pointed to by the links. Link-chasing requests can also specify Level Managers and links so that chains can be constructed (thus allowing a KS to build an overall picture of a solution island). The Level Manager suspends the requesting KS instance and places it in a special waiting state, then sends the requests to the Level Managers specified in the request. When all the LMs in the request have replied, the requesting KS instance is returned to an active state and can be scheduled as usual. When a Level Manager receives a link-chasing request, it searches its local database for the requested information. There is a protocol implemented in the current version of the CASSANDRA shell for handling link-chasing requests.

The important point about the relationship between KSs and channels is that KSs are never given direct access to ports or channels. Instead, they request the containing Level Manager to handle the message on their behalf. In the usual case, the KS instance will be suspended and placed in a waiting state. In the case in which a KS instance wishes to send an entry to another Level Manager, the message containing the entry is passed to the LM and the KS continues processing in the normal way. It is only in the case that a request for communication needs a reply from another Level Manager that KS instances are placed in a waiting state. It should be remembered that once a KS instance has been placed in a wait state, it may be the case that it never gets re-activated: this is because the time

taken for the communication to succeed may be greater than the time required to solve a problem or because the receiving Level Manager has crashed or deadlocked. By making Level Managers responsible for all inter-LM communications, it is possible to implement deadlock (and livelock) recovery schemes: this would not be possible if all communications were handled directly by KSs.

The CASSANDRA architecture is designed to remedy some of the modularity problems inherent in the blackboard architecture. There are two basic areas in which this has been achieved: the first is the Level Manager construction itself, the second is the local control property of Level Managers. Since Level Managers are considered to be wholly independent agents within a problem solving system and since it is assumed that they may not have direct access to the internal states of other Level Managers, it follows that they must be responsible for the control of their own problem solving activity. This entails that they must have a local control component.

The local control component of a CASSANDRA system enables the system builder to define control strategies which are precisely tailored to the needs of each Level Manager in the system. This contrasts with the control problem in the conventional blackboard architecture in which both local and global control regimes are mixed (perhaps this is one reason why schedulers are so difficult to write and why it is so difficult to ensure that they perform optimally). The architecture assumes that there will be some

form of global control in any system. The existence of two levels of control permits control structures to be developed in three phases: define local control needs, define global control needs and integrate local and global control regimes.

The fact that Level Managers in CASSANDRA systems are independent further increases the modularity of any system built using the CASSANDRA architecture. It is often stated that two attractive properties of the blackboard architecture are the facts that Knowledge Sources are independent and may only communicate via the blackboard. In a conventional blackboard system, it is possible to add or remove Ks without impacting upon the system other than changing the quality of the solution it generates. In a CASSANDRA system, on the other hand, there is this modularity plus the modularity of the Level Managers. It is possible to add or remove Level Managers in a system and provide message sinks for missing Level Managers (the current implementation of the CASSANDRA shell performs message sinking automatically). This permits systems to be developed incrementally to a far greater extent than within the more conventional framework.

3. CASSANDRA-II

The architecture has been used as the basis for the CASSANDRA-II airspace monitoring program. CASSANDRA-II observes a sector of controlled airspace and watches for violations of air traffic control rules. The input to the

system is in the form of simulated Mode-S radar returns with some higher-level information included: the higher-level information could be recovered from Mode-S radar by performing some post-processing. The program runs in near real time, even though it is implemented on a slow processor in unoptimised compiled LISP.

The monitoring program is concerned with flight level allocation violations, and with vertical and horizontal separation violations. The program is structured as nine Level Managers and contains approximately fifty Knowledge Sources. The Level Managers are organised in pairs, with the ninth acting as an interface between the input signal and the four pairs of Level Managers. The output from the program is in the form of warning messages: each message refers to the aircraft involved in the violation, the type of violation and the positions and altitudes of the aircraft. Each warning message also contains the time at which the violation was detected. The structure of CASSANDRA-II is shown in fig. 1.

FIGURE 1 HERE

The input LM accepts a sequence of descriptors from the simulator. The task of this LM is to remove from consideration all aircraft outside of the sector being monitored. It also detects crashes, and, should there be any, immediately informs the user (this is so that emergency services can be alerted as soon as possible). The remaining task of this LM are to create messages to be sent to the

other LMs in the system. Message creation involves finding all pairs of aircraft in the monitored sector: air traffic control rules are couched in terms of aircraft pairs. Once the pairs have been found, the horizontal and vertical distances between the aircraft are calculated, and the messages are sent to other Level Managers. In the case of the Cruise Level Allocation LM pair, the messages sent by the input LM is in the form of a list of aircraft descriptors: this is because Cruise Level Allocation is a property of individual aircraft.

The remaining Level Managers in the program are paired. In each case there is an upper and a lower Level Manager. The upper LM is charged with pruning the input message. The pruning consists of removing references to aircraft which cannot possibly participate in a rule violation. For example, the Vertical Separation upper LM removes all pairs of aircraft which are separated by 4000 ft. or more in the vertical and by 5 nautical miles or more in the horizontal. The choice of 4000 ft. is determined by the fact that this figure is the minimum vertical separation for supersonic aircraft above 45,000 ft.; the choice of 5 nautical miles is determined by the basic slot size for all aircraft in controlled airspace.[3] When pruning has been performed, messages are sent to the lower LM of the pair. The lower LM is responsible for performing checks for actual violations

[3] A slot is a box of air 5 nautical miles by 5 nautical miles by 1000 ft.. Slots form the basis of airspace management in the UK.

and for sending warning messages to the user in the case of violations.

Communication between Level Managers in CASSANDRA-II is uni-directional. Messages pass only from the input LM towards the message interface. No messages may flow in the opposite direction. This is a property of the task domain and has the comforting implication that the program is deadlock-free. The direction in which messages flow is shown by the arrows in figure 1. Messages only flow along the channels denoted by the arrows in the figure: no Level Manager may communicate with any other except via a channel.

Each Level Manager in CASSANDRA-II has its own local controller. There are three generic local controllers in the program. The generic controllers were selected on the grounds that they were the most appropriate for the Level Managers which they control. The generic control strategies are:

- sequence,
- LIFO queue, and
- priority queue.

The sequence strategy was introduced in order to more fully examine the properties of local control in the architecture. It is present in all of the upper Level Managers and in the input LM. It could be removed at the cost of increasing the size of the Knowledge Sources in those Level Managers. Sequence is used to schedule KS instances in a particular

order. In the input LM, for example, the KS scheduling sequence is: remove all references to aircraft outside the monitored sector, find all aircraft pairs, compute the horizontal and vertical distances between paired aircraft, check for collisions, send messages. The sequence could be compressed into one large KS, but this would tend to reduce the responsiveness of the LM to messages from other sources. It is a general principle in the CASSANDRA architecture that KSs be small: the KSs in the upper LMs of CASSANDRA-II follow this prescription.

The LIFO regime is used when there is no information upon which to base priority calculations. It is used only in the Cruise Level Allocation lower LM. Cruise Level Allocation is concerned with determining whether or not an aircraft is flying at the correct height for its compass heading. In the current version of CASSANDRA-II, this LM only has the aircraft descriptors to form a basis for decisions. The order in which tasks are scheduled, therefore, is wholly arbitrary. The control stack is based on the order in which messages arrive at the LM.

The final control regime is used whenever there is distance information. Priority queue is used by the vertical, longitudinal and lateral separation lower LMs. The regime assigns a priority which is inversely proportional to the distance between two aircraft: the smaller the distance, the higher the priority.

Local control in CASSANDRA-II is tailored to the task which each LM must perform. Global control in the program is designed to ensure that each LM has an equal share of processor time (the system is implemented on a uni-processor system). In CASSANDRA-II, it matters little which LM is scheduled at one time: this is because they are considered to be operating in parallel. The one constraint which is imposed on the system of Level Managers is that the input LM must be activated first so that data can be obtained from the airspace simulation. The current global scheduler operates an infinite loop in which it first activates the input LM, then activates the Cruise Level Allocation LM pair, Vertical Separation LM pair, Longitudinal Separation LM pair and finally the Lateral Separation LM pair. When a pair is activated, the upper LM is activated before the lower LM so that pruning can be performed and messages sent to the lower LM.

4. HASP/SIAP

The global control regime in the CASSANDRA-II system is extremely simple. This simplicity is due to the nature of the task domain in which the system operates. In order to determine the applicability of the CASSANDRA architecture, it was decided to try to re-design the HASP/SIAP system ([Nii, 1986b]; [Feigenbaum, 1983]; [Feigenbaum, 1978]) as a CASSANDRA system. The most interesting point in this study was the development of a global scheduler which would reflect the control strategy used by HASP/SIAP. This section

is concerned with a brief description of the resulting system. The reader should consult ([Craig, 1987], ch. 6) for more details.

In the CASSANDRA version of HASP/SIAP, each blackboard level of the original system is represented as a complete Level Manager. The Level Managers in the CASSANDRA version (called HASP-C) contain all the entries which would be found on the corresponding level of the HASP/SIAP blackboard. Each HASP-C LM also contains the KSs which trigger in response to events in the local database of that LM. HASP-C Level Managers also contain a local controller.

The kernel of the HASP/SIAP system is the control mechanism. The control mechanism is responsible, basically, for maintaining three control lists: an event list, an expected event list and a clock event list. The first list records blackboard events which may be of interest to KSs in the system; the second contains references to events which are expected to happen, given the events which have gone before; the clock event list records times at which KSs are to be scheduled. The HASP/SIAP scheduler examines the control lists and executes all KSs which have responded to events. The event lists are processed in the following order: expected events, blackboard events and clock events. When an event list is selected, all the events it contains are processed. The termination condition of each event list sub-cycle is that all the events in the list at the start of the sub-cycle should have been dealt with.

In HASP-C, each Level Manager has its own set of event lists. The event lists in a Level Manager record the events which have taken place within that particular LM. The local controllers in all the LMs of HASP-C are identical: that is, they adopt the same strategy. The global controller is charged with sending control messages on a cyclic basis to all Level Managers in the system. Control messages instruct each LM to process the contents of a given event list. The global controller implements the loop described above: it first sends messages instructing each LM to attend to its expected events, then messages instructing LMs to attend to simple events and, finally, messages instructing each LM to process its clock events.

When a Level Manager receives a control message, it checks the specified event list. If the list is empty, the LM ignores the message. If there are events in the list, it executes the Knowledge Sources which have triggered on those events. Knowledge Source execution causes new events and causes messages to be sent to other Level Managers in the system.

The role of the global scheduler in HASP-C is to provide a coarse synchronisation between the Level Managers in the system. The HASP/SIAP system is roughly pipelined and is, consequently, driven by the availability of information across levels: the structure of the HASP-C system is similar. Because of the distributed nature of HASP-C, it is not possible for the global controller to enforce strict synchronisation between Level Managers: instead, it keeps

the LMs performing the basic control cycle, even though they may be out of synchronisation. Since the system is driven by data availability, this ensures that tasks will be performed in the same order across the system.

5. CONCLUSIONS

The CASSANDRA architecture has been briefly described. It is based on the blackboard model of problem solving, but differs in its increased modularity and by the fact that it admits of a naturally distributed implementation. The architecture has been used to build a real time monitoring program, CASSANDRA-II, which monitors air traffic control rule violations. The architecture has also been used as the basis of a re-design of the HASP/SIAP sonar interpretation system. It is believed that the near real-time performance of the CASSANDRA-II system is directly attributable to the architecture.

6. REFERENCES

[Balzer, 1980] Balzer, R., Erman, L., London, P. and Williams, C. HEARSAY-III: A Domain Independent Framework for Expert Systems. Proc. First Annual Conference on Artificial Intelligence, pp. 108 - 110, 1980

[Craig, 1986] Craig, I.D. The Ariadne-1 Blackboard System Computer Journal, Vol. 29, No. 3, pp. 235 - 240, 1986

[Craig, 1987] Craig, I.D. Decentralised Control in a Blackboard System, Ph.D. Thesis, Department of Computing, University of Lancaster, Lancaster, UK, March, 1987

[Erman, 1975] Erman, L.D. and Lesser, V.R.,
A Multi-level Organisation for Problem Solving Using Many,
Diverse, Cooperating Sources of Knowledge.
Proc. IJCAI 4, Vol. 2, pp. 483 - 490, 1975

[Erman, 1981] Erman, L.D., London, P. and Fickas, S.
The Design and an Example Use of HEARSAY-III.
Proc. IJCAI 7, Vol. 1, pp. 409 - 415, 1981

[Feigenbaum, 1978] Feigenbaum, E. and Nii, H.P.,
Rule-based Understanding of Signals.
Pattern -Directed Inference Systems, D.A. Waterman
and F. Hayes-Roth (Eds.), Academic Press, 1978

[Feigenbaum, 1982] Feigenbaum, E., Nii, H.P.,
Anton, J.J. and Rockmore, A.J.
Signal-to-signal transformation: HASP/SIAP case
study
AI Magazine, Vol. 3, pp 23 - 35, 1982

[Hayes-Roth, 1979] Hayes-Roth, B. and Hayes-Roth, F.,
A Cognitive Model of Planning, Cognitive Science,
Vol. 3., pp. 275 - 310, 1979

[Hayes-Roth, 1983] Hayes-Roth, B. The Blackboard Architecture:
A General Framework for Problem Solving?
Report No. HPP-83-30, Heuristic Programming
Project, Computer Science Dept.,
Stanford University, Palo Alto, CA, May 1983

[Hayes-Roth, 1984] Hayes-Roth, B. A Blackboard Model of Control.
Report No. HPP-83-38, Heuristic Programming Project,
Computer Science Department,
Stanford University, Palo Alto, CA.,
June 1983 (Revised December, 1984)

[Hayes-Roth, 1984b] Hayes-Roth, B. BB-1: An Architecture
for blackboard systems that control, explain, and learn
about their own behavior.
Technical Report HPP-84-16, Stanford University, 1984

[Hayes-Roth, 1985a] Hayes-Roth, B and Hewett, M.
Learning Control Heuristics in BB-1
Technical Report HPP-85-2,
Stanford University, 1985

[Hayes-Roth, 1985b] Hayes-Roth, B.
A Blackboard Model for Control
Artificial Intelligence, Vol. 26, pp. 251 - 322, 1985

[Hayes-Roth, 1986] Hayes-Roth, B., Garvey, A.,
Johnson, M.V. and Hewett, M.
A Layered Environment for Reasoning about Action,
Technical Report No. KSL 86-38,
Knowledge Systems Laboratory,
Stanford University, 1986

[Nii, 1979] Nii, H.P. and Aiello, N. AGE:
A knowledge-based program for buidling knowledge-based programs.
Proc. IJCAI 6, pp. 645 - 655

[Nii, 1986a] Nii, H.P.
~~The Blackboard Model of Problem Solving~~
Artificial Intelligence Magazine,
Vol. 7, No. 2, pp. 38 - 53, 1986

[Nii, 1986b] Nii, H.P.
Blackboard Systems Part Two: Blackboard Application Systems
Artificial Intelligence Magazine,
Vol. 7, No. 3, pp. 82 - 106, 1986

[Terry, 1983] Terry, A. The CHRYSALIS Project:
Hierarchical Control of Production Systems. Memo HPP-83-19,
Heuristic Programming Project,
Computer Science Dept., Stanford University,
Palo Alto, CA., May, 1983

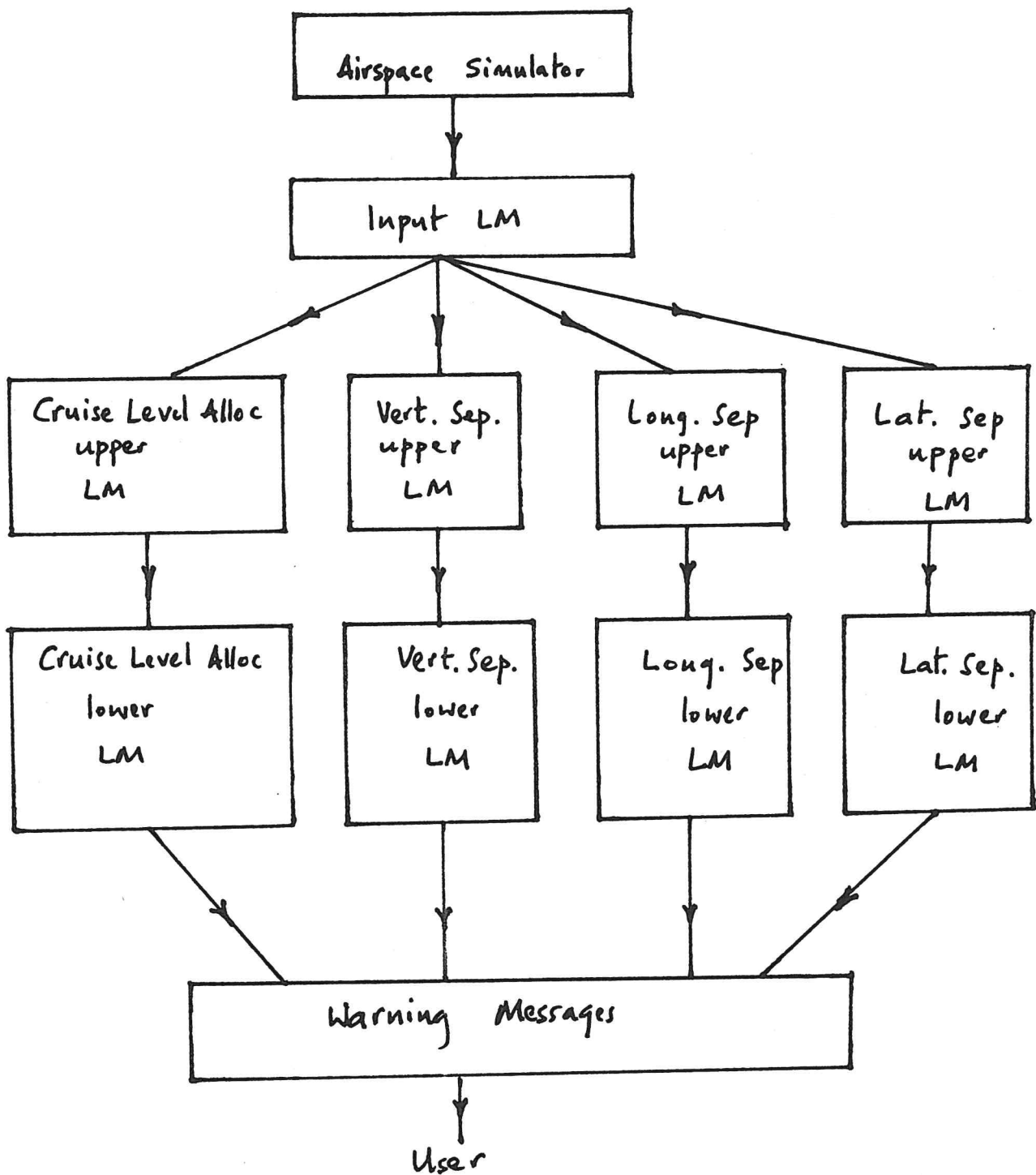


Figure 1

